Attorney's Docket No. 0559-596001 / P21148

APPLICATION FOR UNITED STATES LETTERS PATENT

in the name of

William R. Wheeler,
Matthew J. Adiletta,
Christopher Clark
and
Timothy J. Fennel

of

INTEL CORPORATION

for

MODEL-BASED LOGIC DESIGN

Kenneth F. Kozik
Fish & Richardson P.C.
225 Franklin Street
Boston, Massachusetts 02110
Tel.: (617) 542-5070
Fax: (617) 542-8906

CERTIFICATE OF MAILING BY EXPRESS MAIL

Express Mail Label No. EL 624 273 699 US

I hereby certify that this correspondence is being deposited with the United States Postal Service as Express Mail Post Office to Addressee with sufficient postage on the date indicated below and is addressed to the Commissioner for Patents, Washington, D.C. 20231.

August 28, 2001
Date of Deposit

Samantha Bell
Signature

Samantha Bell
Typed or Printed Name of Person Signing Certificate

MODEL-BASED LOGIC DESIGN

TECHNICAL FIELD

This invention relates to model-based logic design.

BACKGROUND

Microprocessor logic design typically includes an architectural stage and an implementation stage.

The architectural stage includes designing a framework of functional units that provide performance and functionality of a new microprocessor. This framework is typically captured in a text-based document. A model of the new microprocessor, represented in a high level language such as C++, is generated to verify that the function and performance requirements are met.

The implementation stage involves taking the model and the text-based document from the architectural stage and generating a Hardware Design Language (HDL) file.

DESCRIPTION OF DRAWINGS

FIG. 1 shows a system.

FIG. 2 shows a graphical model.

FIG. 3 shows a data structure.

FIG. 4 shows a logic modeling process.

DETAILED DESCRIPTION

Referring to FIG. 1, a system 10 includes a computer 12, such as a personal computer (PC). The computer 12 may be connected to a network 14, such as the Internet, that runs TCP/IP

(Transmission Control Protocol/Internet Protocol) or another suitable protocol. Connections may be via Ethernet, wireless link, or telephone line.

The computer 12 contains a processor 16 and a memory 18. Memory 18 stores an operating system (O/S) 20 such as Windows2000® or Linux, a TCP/IP protocol stack 22 for communicating over network 14, and machine-executable instructions 24 executed by processor 16 to perform a logic modeling process 100 below. The computer 12 also includes an input/output (I/O) device 26 for display of a graphical user interface (GUI) 28 to a user 30 and a storage device 32 for storing a database 34.

The logic modeling process 100 executes in the computer 12. The logic modeling process 100 is a process in which a C++ model file and a Hardware Design Language (HDL) file, such as a Verilog (IEEE Standard 1364) file or a Very high speed integrated circuit Hardware Design Language (VHDL) (IEEE Standard 1076) file, are generated from a successively refined graphical model. Verilog is a hardware description language, a textual format for describing electronic circuits and systems, applied to electronic design. Verilog is used for verification through simulation, for timing analysis, for test analysis (testability analysis and fault grading) and for logic synthesis.

Referring to FIG. 2, a graphical model 50 is a pictorial model of a microprocessor design. The graphical model 50 is a tool that blends the use of textual description and graphical description of logical elements to hierarchically capture a

silicon design. The use of graphics is important to "re-use" capability and for design support. The graphical model 50 is represented in database 34 to enable production of a cycle-accurate high performance simulation model as well as a synthesizable Verilog model.

The graphical model 50 includes of a set of blocks 52, interconnected by lines 54. Each of the blocks 52 represents logical elements of a device under design. The lines 54 represent connections of block inputs to block outputs. Every block in the graphical model 50 is an instance of a specific type of block. The type of block determines the relationship between a block's outputs and its inputs, states, and time. The graphical model 50 may contain any number of instances of any type of block needed to model a system.

Although not illustrated in FIG. 2, each of the blocks 52 and lines 54 are linked to a Register Transfer Diagram (RTD) descriptions allowing the user 30 to navigate and drill down to a particular place in the design quickly. This allows the user 30 to capture the design intent and then to successively refine the design. In a RTD view (not shown) of the graphical model 50 logic is color-coded. For example, state elements are shown in blue, semantically correct combinatorial logic is shown in green, common blocks describing pipe stages are shown in white, ports identifying inputs/outputs of the RTD are shown in yellow, and library elements with correctly matched inputs and outputs are shown in gray. Other schemes can also be used.

Each of the blocks 52 and corresponding lines 54 are stored in the database 34. More specifically, the blocks and corresponding connections are stored in one or more data structures that represent the gates, nodes and nets of the device. The data structures provide an internal list or description of a net list of the device is stored in the database 34.

Referring to FIG. 3, an exemplary data structure 60 implemented in C++ and stored in the database 34 and representing an exemplary description of a graphic model includes M-Gate (62), M_Pin (64), M_Net (66) and M_node (68), where ``M'' represents Model. The data structure may be implemented, for example, as a linked list or binary tree.

The M-Gate (62) represents a logical function, for example, AND, FF, FIFO, and so forth. The M_Gate (62) contains zero or more input pins (M_pin's) and zero or more output pins. Each distinct logical function in the graphical model 50 is assigned its own derived M_gate type.

The M_pin (64) represents a connection point to a gate. The connection may be either an input or output. It connects an M_gate to an M_net.

The M_node (66) represents the total extent (i.e. all bits) of a simulation state.

The M_Net (68) represents an arbitrary collection of bits within an M_node. An M_net connects one or more M_pin's together.

M-Gate (62), M_Pin (64), M_Net (66) and M_node (68) are all C++ classes in which there are multiple derived classes for each class listed above.

The data structure 60 is updated and reflects the current state of the graphical model 50. The data structure 60 is used by the logic modeling process 100 to generate an architectural model using, for example, C++ constraints, and a Verilog implementation model. Thus, a single database, i.e., the database 34, is used as generator of both an architectural and implementation model for a chip design.

Referring to FIG. 4, the logic modeling process 100 includes graphically capturing (102) combinatorial blocks, state elements and graphical library elements. Graphics elements in the library may be definable or may have predefined functions. For example, the library may contain software objects that perform the function of a FF or a latch. The library may also contain graphics elements that are undefined, i.e., that have no code associated with them.

A block in a graphical model may represent a single combinatorial element, such as a multiplexer or state element. A combinatorial block represents the functionality of several combinatorial elements or the function of several state elements.

The process 100 builds (104) a control/design analysis and checks (106) for errors. For example, process 100 determines if there are any un-terminated or inconsistent connections in the design. If any such problems are detected, process 100 issues an error message to the logic designer. The error message may

specify the nature of the problem and its location within the logic design. The logic designer is then given the opportunity to correct the problem before process 100 moves forward.

The process 100 stores (108) the logic of the graphics capture in a data structure. The process 100 determines (110) whether to write an architectural model or implementation model. If an implementation model is to be written, the process 100 generates (112) Verilog using Verilog constructs to provide the implementation model.

If an architectural model is to be written, the process 100 coverts (114) the data structure into a C++ topology. The process provides (116) timing and clock domain assignments partitions (118) clock domain topologies. Each partition is coded ordered (120) and partition code provided to a C++ compiler.

Code ordering means that the logical constructs are sorted based on producer/consumer. BY subsequently code-ordering the C++ model may be simulated as a single call model. A single call model means that each logical construct is evaluated only once per cycle. Hence, the C++ model simulator is a cycle-based simulator. The Verilog model is also written after being extracted from the data structure and is typically simulated using an event driven simulator such as ModelSim™ from Model Technology, for example.

The process writes (122) C++ files, batch files and makefiles for the architectural model. Thus, process 100

generates high performance C++ and highly efficient Verilog from the same database.

Process 100 may be implemented using one or more computer programs executing on programmable computers that each includes a processor, a storage medium readable by the processor (including volatile and non-volatile memory and/or storage elements), at least one input device, and one or more output devices.

Each such program may be implemented in a high level procedural or object-oriented programming language to communicate with a computer system. Also, the programs can be implemented in assembly or machine language. The language may be a compiled or an interpreted language.

Each computer program may be stored on a storage medium or device (e.g., CD-ROM, hard disk, or magnetic diskette) that is readable by a general or special purpose programmable computer for configuring and operating the computer when the storage medium or device is read by the computer to perform process 100.

Process 100 may also be implemented as a computer-readable storage medium, configured with a computer program, where, upon execution, instructions in the computer program cause the computer to operate in accordance with process 50.

Further aspects, features and advantages will become apparent from the following.